INFORME PRÁCTICA ALGORITMO BACKTRACKING

(PRÁCTICA NÚMERO CUATRO)

HÉCTOR ALBERCA SÁNCHEZ-QUINTANAR
DARIO ANDRÉS FALLAVOLLITA FIGUEROA

Índice

1. Elementos del problema relacionándolos con el algoritmo de backtracking planteado.

Tenemos un vector que contiene 7 números del 0 a 9 y 2 signos aleatorios entre (“+”, “-“, “/”, “*”). En cada ejecución cambian tanto los números como los signos. Para ello utilizamos el método Collections.shuffle(). A parte de este vector de números y operadores disponibles, el método backtracking contiene un array de caracteres, el array estado. En este array vamos a ir añadiendo los elementos y a comprobar si es el vector solución que buscamos. También tenemos un entero, la etapa, que nos indica en que etapa del algoritmo estamos. En nuestro caso, en cada etapa se añade un número u operador al vector estado, comprobando así si es nuestra solución o no. Por último, el método toma un parámetro resultado, que pasamos al método *esSolucion()* y que se genera aleatoriamente. El método utilizado es el siguiente:

```
public void backtracking(int resultado, int etapa, char[] vectorDisponibles, char[] estado) {
    char[] descent;

    if (etapa == 5 && !enc) {
        if (enc = esSolucion(resultado, etapa, estado)) {
            procesarSolucion(resultado, estado);

        }
    } else if (!enc && etapa < estado.length) {
        descent = generarDescendientes(vectorDisponibles, etapa);

        for (int i = 0; i < descent.length; i++) {
            estado[etapa] = descent[i];
            backtracking(resultado, etapa + 1, vectorDisponibles, estado);

        }
    }
}
```

2. Complejidad del algoritmo

La complejidad de este algoritmo revoca en la recursividad. Hemos de tener en cuenta el método backtracking() así como todos los métodos contenidos en este.

Los elementos que tenemos que considerar son:

- ➔ Elementos por etapa: x.
- ➔ Tenemos una complejidad lineal para generar los descendientes (O(n)).
- ➔ En el método esSolucion() la complejidad es despreciable.
- ➔ En el método procesarSolucion() también podemos despreciar la complejidad.

Teniendo en cuenta: O(1) + kT(n-1) llegamos a la conclusión de que la complejidad es $O(k^n)$.